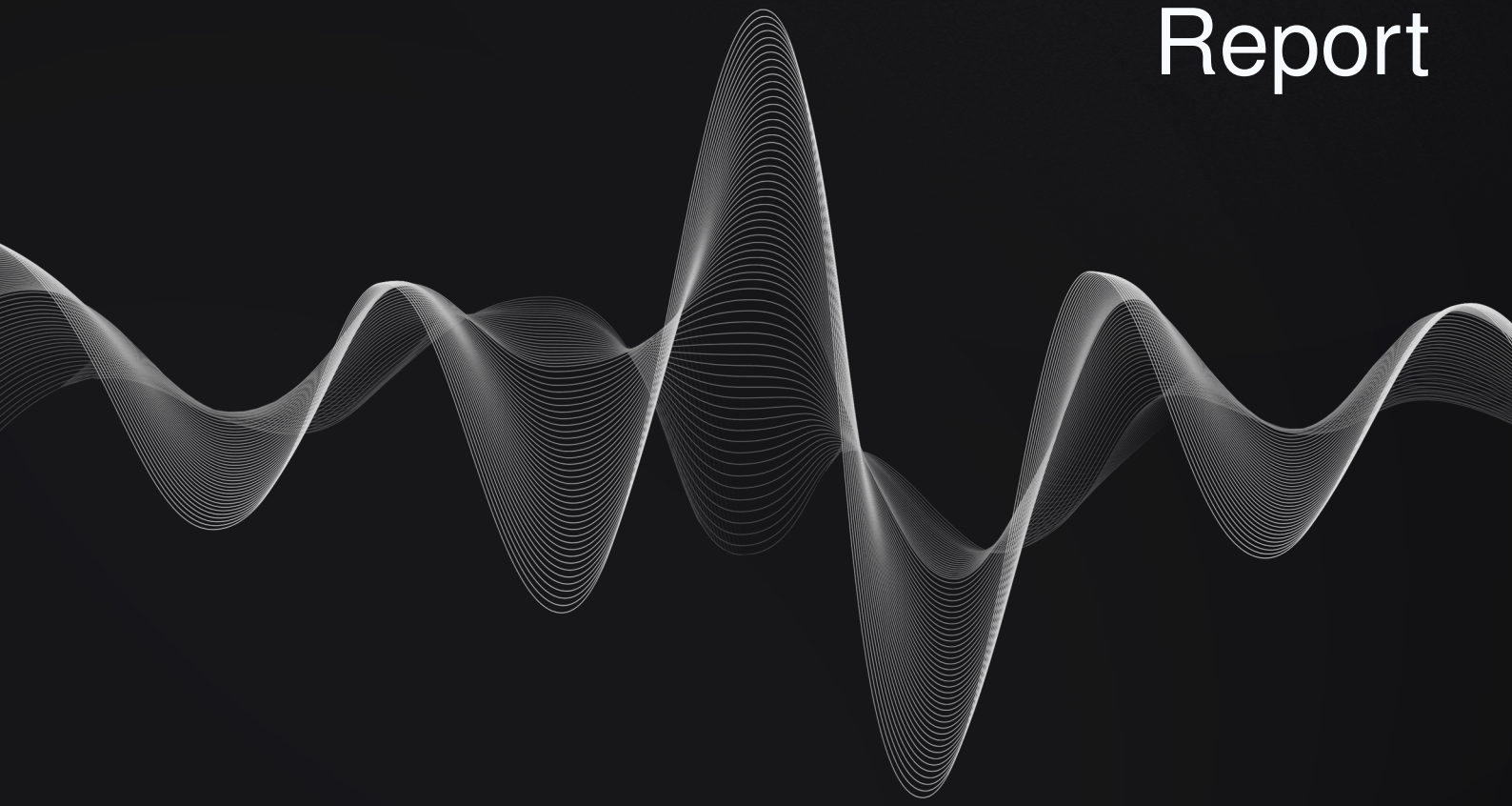
RESONANCE

Blend

Blend Foundation

Blend Protocol CosmWasm Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.0)

Audit_Report_BLEND-PRO_FINAL_20

Jul 25, 2023	v0.1	Michał Bazyli: Initial draft
Aug 4, 2023	v0.2	Michał Bazyli: Added findings
Aug 7, 2023	v1.0	Charles Dray: Approved
Aug 14, 2023	v1.1	Michał Bazyli: Reviewed findings
Aug 14, 2023	v1.2	Michał Bazyli: Finalized report
Aug 14, 2023	v2.0	Charles Dray: Published

Points of Contact	Theo	Blend Foundation	theo@blend.foundation
	Charles Dray	Resonance	charles@resonance.security
	Luis Lubeck	Resonance	luis@resonance.security
Testing Team	Michał Bazyli	Resonance	michal.bazyli@resonance.security
	Ilan Abitbol	Resonance	ilan.abitbol@resonance.security
	João Simões	Resonance	joao.simoes@resonance.security

Copyright and Disclaimer

Contents

1 Document Control 2

Copyright and Disclaimer 2

2 Executive Summary 5

System Overview 5

Repository Coverage and Quality 5

3 Target 6

4 Methodology 7

Severity Rating 8

Repository Coverage and Quality Rating 9

5 Findings 10

Non-Withdrawable Surplus Asset in ghost 12

Potential Panic during deposit_funds function Invocation in funds-router 13

Denom order dependecy 14

Migrating contracts without state migrations may cause problems 15

save_router_address will always Error due to Uninitialized Data 16

ghost_markets are not saved during the instantiation of fund-factory 17

Missing validation of parameters during instantiate in bow contract 18

Possible storage bloating of BOW_CONTRACTS in bow contract 19

Missing parameter validation when updating the bow config 20

Missing parameter validation when updating fin config 21

Missing validation of parameters during instantiate in fin contract 22

Possible storage bloating of GHOST_MARKETS in ghost contract 23

Lack of fee validation 24

Inefficient update of CLAIM_DENOMS 25

Redundand ADMIN check in fund-factory 26

Redundand ADMIN check in oracle update 27

Redundant string conversion 28

Queries might reach gas limit 29

Events are not emitted when executing essential operations 30

Redundant addr validation 31

Lack of Error Handling 32

Two-step ownership transfer is missing 33

Confusing function naming 34

Confusing variable naming 35
Usage of nonconventional CosmWasm msg 36
Usage unwrap_or_default 37
Lack of default slippage if not provided 38

A Proof of Concepts **39**

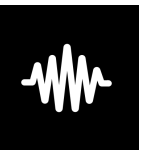
Executive Summary

Blend Foundation contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between July 24, 2023 and August 04, 2023.The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 10 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Blend Foundation with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

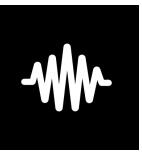
The audit process unveiled a significant number of issues with the existing codebase, most prominently related to protocol instantiation and the inability to fully test the protocol's operation. These fundamental issues have prevented a thorough and meaningful evaluation of the complete code flow, leaving potential vulnerabilities and bugs undetected. It's imperative to address these foundational problems promptly and rigorously. Post remediation, we strongly advise scheduling a re-audit to ensure that the implemented fixes effectively resolve the issues and that the protocol operates as expected in a secure and efficient manner. The re-audit will further contribute to enhancing the protocol's reliability, efficiency, and overall security.



System Overview

The **Blend Protocol** is a protocol built on Rust CosmWasm that enables decentralized strategy ideation and creation.

This system provides interfaces for seamless interaction with FIN, ORCA, GHOST, and BOW, all integral components of the Kujira Layer-1 Blockchain.



Repository Coverage and Quality

This section of the report has been concealed by the request of the customer.

Target

The following items are included as targets of the security assessment:

- Repository: REDACTED
- Hash: ba1405dd8b608967650d3b41023059773760c2dd

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial-related attack vectors

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Rust CosmWasm

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust CosmWasm audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Frontrunning attacks
- Unsafe third party integrations
- Denial of service
- Access control issues
- Inaccurate business logic implementations
- Incorrect gas usage

- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

Impact \ Likelihood	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
- **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Non-Withdrawable Surplus Asset in ghost		Resolved
RES-02	Potential Panic during deposit_funds function Invocation in funds-router		Resolved
RES-03	Denom order dependecy		Resolved
RES-04	Migrating contracts without state migrations may cause problems		Acknowledged
RES-05	save_router_address will always Error due to Uninitialized Data		Resolved
RES-06	ghost_markets are not saved during the instantiation of fund-factory		Resolved
RES-07	Missing validation of parameters during instantiate in bow contract		Resolved
RES-08	Possible storage bloating of BOW_CONTRACTS in bow contract		Resolved
RES-09	Missing parameter validation when updating the bow config		Resolved
RES-10	Missing parameter validation when updating fin config		Resolved
RES-11	Missing validation of parameters during instantiate in fin contract		Resolved

RES-12	Possible storage bloating of GHOST_MARKETS in ghost contract		Resolved
RES-13	Lack of fee validation		Resolved
RES-14	Inefficient update of CLAIM_DENOMS		Acknowledged
RES-15	Redundand ADMIN check in fund-factory		Resolved
RES-16	Redundand ADMIN check in oracle update		Resolved
RES-17	Redundant string conversion		Acknowledged
RES-18	Queries might reach gas limit		Acknowledged
RES-19	Events are not emitted when executing essential operations		Resolved
RES-20	Redundant addr validation		Acknowledged
RES-21	Lack of Error Handling		Acknowledged
RES-22	Two-step ownership transfer is missing		Resolved
RES-23	Confusing function naming		Acknowledged
RES-24	Confusing variable naming		Acknowledged
RES-25	Usage of nonconventional CosmWasm msg		Acknowledged
RES-26	Usage unwrap_or_default		Resolved
RES-27	Lack of default slippage if not provided		Acknowledged

Non-Withdrawable Surplus Asset in ghost

High | RES-BLEND-PRO01 | Business Logic | Resolved

Code Section

- Not specified

Description

The Ghost contract allows the `FUND` address to borrow assets from Kujira GHOST. When the `FUND` repays the borrowed amount with more assets than required, the surplus assets are sent back to the Ghost contract. However, these returned assets will not be withdrawable by `FUND` due to the absence of a functionality supporting this operation in the contract.

Recommendation

It is advisable to incorporate a mechanism that guarantees precision in withdraws made by FUND, thereby avoiding any surplus. Alternatively, it would be beneficial to implement a function that permits the withdrawal of any surplus assets that are returned to the Ghost contract.

Status

The issue has been fixed in f073d3c1874b769691e9bf04e3497a0f0fe91f22.

Potential Panic during deposit_funds function Invocation in funds-router

High | RES-BLEND-PRO02 | Code Quality | Resolved

Code Section

- contracts/fund-router/src/execute.rs#L134-138

Description

The `deposit_funds` function within the `funds-router` contract permits whitelisted addresses to deposit assets into the FUND in return for the contract token. The quantity of tokens minted is determined by the current position values and the existing supply of the contract token. Nonetheless, if the contract supply is nonzero and the value derived from the current position amounts to zero, the `deposit_funds` function will result in a panic.

Recommendation

It is recommended to reimplement the logic of the function to validate the state of the "contract" denom supply and the current position value. If conditions are detected that would lead to a panic, pre-emptively handle them.

Status

The issue has been fixed in dca2004bd6fb4f9adeebc225b9e9be8dd3c06ba2.

Denom order dependecy

High | RES-BLEND-PRO03 | Denial of Service | Resolved

Code Section

- contracts/bow/src/contract.rs#L96
- contracts/bow/src/contract.rs#L162

Description

The `store_bow_contracts` function in the `bow contract` retrieves the configuration from the Kujira BOW and then extracts the denoms. Following this, the Kujira BOW address is saved in `BOW_CONTRACTS` using a key produced by the `get_key` function.
The `get_key` function creates storage keys by utilizing two denoms in the order they appear in the config. However, it doesn't ensure that these denoms are arranged alphabetically. If the denoms are provided out of sequence, it can lead to incorrect data retrieval from storage during asset deposits. This inconsistency emerges since the denoms transmitted in CosmWasm are always in alphabetical order.

For instance, the denoms retrieved from the `ConfigResponse` of the KujiraBow contract on both testnet and mainnet aren't always in alphabetical order. Therefore, running the deposit function will consistently result in a `ContractError`.

Recommendation

It is advised to introduce a mechanism that will organize the denoms in alphabetical order before storing them in `BOW_CONTRACTS`

Status

The issue has been fixed in a2ffec31b42985e3908e828f17adb59c589f14a5.

Migrating contracts without state migrations may cause problems

High	RES-BLEND-PRO04	Business Logic	Acknowledged

Code Section

- Not specified

Description

The contracts within the scope do not have implemented state migration logic which could lead to inconsistency in the protocol after migration.

Recommendation

It is recommended to remove migration functionality or implement appropiatie logic which follow best migration patters.

Status

The issue has been partially fixed in a0d5ce499fb7b76e59a34ad93ddacdfcee28f5e9 as it is only a placeholder. The development team stated "This finding will be fixed in a future iteration".

save_router_address will always Error due to Uninitialized Data

High | RES-BLEND-PRO05 | Code Quality | Resolved

Code Section

- contracts/fund-factory/src/handlers/save_router.rs#L26

Description

The `save_router_address` function saves router address on reply. However the `save_router_address` function has a critical dependency on a specific piece of data being initialized before its invocation. This dependency is associated with the `FUNDS.update` method call, where the function assumes that the required data is already initialized.

Recommendation

It is recommended to implement a validation step before the `FUNDS.update` call to ensure that the required data has been properly initialized.

If the data is not initialized, handle the condition gracefully, either by initializing the data at that point or returning an informative error message to guide further action.

Status

The issue has been fixed in 84a2bd4eddbeab4b9924fb47655e399bc1231a4c.

ghost_markets are not saved during the instantiation of fund-factory

Medium | RES-BLEND-PRO06 | Code Quality | Resolved

Code Section

- contracts/fund-factory/src/contract.rs#L28-L52

Description

The `instantiate` function in the `fund-factory` contract receives all the necessary parameters to initialize the entire protocol. However, the `ghost_markets` from `InstantiateMsg` is not saved in the `GHOST_MARKETS` . This omission is critical, as it is required to successfully execute the `create_router` function.

Recommendation

It is recommended to update the `instantiate` function in the `fund-factory` contract to ensure that the `ghost_markets` from `InstantiateMsg` is properly saved in the `GHOST_MARKETS` constant. This adjustment is essential for the correct execution of the `create_router` function within the protocol, ensuring consistency and preventing potential issues in the process.

Status

The issue has been fixed in 24c6d74ae3ccf38b50bd6c0234d887e0af24cdb6.

Missing validation of parameters during instantiate in bow contract

Medium | RES-BLEND-PRO07 | Data Validation | Resolved

Code Section

- contracts/bow/src/contract.rs#L50
- contracts/bow/src/contract.rs#L54

Description

During the contract's instantiation phase, some values are not verified, and only a check for the correct address is performed. This lack of validation could result in potential inconsistencies within the protocol if incorrect values are supplied.

- bow_staking
- claim_denoms

Recommendation

It is recommended adding validation checks for all crucial parameters when instantiating the contract. This will help prevent potential inconsistencies or errors.

Status

The issue has been fixed in69be8ac60d097934c9752fa11374588df9c9fd49 .

Possible storage bloating of BOW_CONTRACTS in bow contract

Medium | RES-BLEND-PRO08 | Denial of Service | Resolved

Code Section

- contracts/bow/src/contract.rs#L97

Description

The `store_bow_contracts` function called durint instantiation of the contract or during execution `update_config` function allows the ADMIN to update the `BOW_CONTRACTS` contract addresses. However, there is no functionality to remove unused or old contracts which can result in storage bloating.

Recommendation

It is recommended to develop a mechanism that facilitates the removal of outdated or no longer needed data from the storage.

Status

The issue has been fixed in ca81a3af3a6bfd9bfb5fb6fa85155c192bbf5a10.

Missing parameter validation when updating the bow config

Medium | RES-BLEND-PRO09 | Data Validation | Resolved

Code Section

- contracts/bow/src/contract.rs#L125
- contracts/bow/src/contract.rs#L129

Description

The `update_config` function permits the `ADMIN` to modify the configuration. Nevertheless, certain parameters are not undergoing validation checks, which could potentially result in protocol inconsistencies:

- bow_staking
- claim_denoms

Recommendation

It is recommended to include validation checks for all crucial parameters when modifying the protocol's configuration. However, the optimal approach would be to eliminate the ability for an `ADMIN` to update these parameters and ensure all validation occurs during the contract's instantiation. This method will aid in avoiding possible inconsistencies or mistakes.

Status

The issue has been fixed in 69be8ac60d097934c9752fa11374588df9c9fd49 and f9839b9bb5ba29c45f5da9be6ff0ffaa462246be.

Missing parameter validation when updating fin config

Medium | RES-BLEND-PRO10 | Data Validation | Resolved

Code Section

- contracts/fin/src/contract.rs#L83-89

Description

The `ADMIN` has the ability to modify the configuration of the fin contract. Nevertheless, certain parameters are not undergoing validation checks, which could potentially result in protocol inconsistencies.

- SWAP_PATHS

Recommendation

It is recommended to include validation checks for all crucial parameters when modifying the protocol's configuration. Additionally, a validation path should be added to ensure that swap paths exist. However, the optimal approach would be to eliminate the ability for an `ADMIN` to update these parameters and ensure all validation occurs during the contract's instantiation. This method will aid in avoiding possible inconsistencies or mistakes.

Status

The issue has been fixed in 7e9b485960aa1d547e9f8009be2bac0184db0658.

Missing validation of parameters during instantiate in fin contract

Medium | RES-BLEND-PRO11 | Data Validation | Resolved

Code Section

- contracts/fin/src/contract.rs#L61

Description

During the contract's instantiation phase, some values are not verified, and only a check for the correct address is performed. This lack of validation could result in potential inconsistencies within the protocol if incorrect values are supplied.

- swap_paths

Recommendation

It is recommended to add validation checks for all crucial parameters, including verifying if swap paths exist, when instantiating the contract. This additional scrutiny will help prevent potential inconsistencies or errors.

Status

The issue has been fixed in e9b485960aa1d547e9f8009be2bac0184db0658.

Possible storage bloating of GHOST_MARKETS in ghost contract

Medium | RES-BLEND-PRO12 | Code Quality | Resolved

Code Section

- contracts/ghost/src/contract.rs#L129

Description

The `store_ghost_markets`function called in `update_config` allow the `ADMIN` to update the `GHOST_MARKETS` contract addresses. However there is no functionality to remove unused or old contract which can result in storage bloating.

Recommendation

It is recommended to develop a mechanism that facilitates the removal of outdated or no longer needed data from the storage.

Status

The issue has been fixed in 7e2b743c0e8735e6cba6d0973951fc5ed3b4d7cb.

Lack of fee validation

Medium | RES-BLEND-PRO13 | Data Validation | Resolved

Code Section

- contracts/fund-router/src/contract.rs#L42

Description

The `fee` value has not been confirmed to fall within safe range values during the instantiation of the contract or updating the config. This could result in incorrect fee calculations within the protocol.

Recommendation

It is recommended to introduce a validation routine that ensures the fee value falls within a safe range of values.

Status

The issue has been fixed in 9586d4a5f1be5f0dde104a9991618a39a1aef725.

Inefficient update of CLAIM_DENOMS

Low | **RES-BLEND-PRO14** | Code Quality | **Acknowledged**

Code Section

- contracts/bow/src/contract.rs#L129

Description

Current implementation of `update_config` allow `ADMIN` to update the `CLAIM_DENOMS`. However `CLAIM_DENOMS.save(deps.storage, &claim_denoms)?;` code will overwrite the current array of denoms instead of updating it.

Recommendation

It is recommended using a method of appending data rather than overwriting for increased flexibility. Alternatively, consider employing the `update` function as opposed to the `save` method.

Status

The issue was acknowledged by Blend's team. The development team stated "It is less error-prone to provide a full consistent list that can be collected and verified off-chain, than updating it on-chain and tracking consistency.".

Redundand ADMIN check in fund-factory

Low | **RES-BLEND-PRO15** | Gas Optimization | **Resolved**

Code Section

- contracts/fund-core/src/contract.rs#L90
- contracts/fund-core/src/contract.rs#L102

Description

A redundant `ADMIN` check exists when updating the fund-core contract. The first check occurs in `contracts/fund-core/src/contract.rs#L90`, and a second one happens within the update function at `contracts/fund-core/src/contract.rs#L102`.

Recommendation

It's recommended to streamline the process by eliminating the redundant ADMIN check when updating the oracle configuration. Maintaining only one check either in `contracts/fund-core/src/contract.rs#L90` or within the update function at `contracts/fund-core/src/contract.rs#L102` should suffice, improving the code's efficiency and readability.

Status

The issue has been fixed in e6983ef20089f1a75dbe99f5c32be0deafd4fdc5.

Redundand ADMIN check in oracle update

Low | **RES-BLEND-PRO16** | Gas Optimization | **Resolved**

Code Section

- contracts/oracle/src/contract.rs#L44
- contracts/oracle/src/contract.rs#L51

Description

A redundant `ADMIN` check exists when updating the oracle contract. The first check occurs in `contracts/oracle/src/contract.rs#L44`, and a second one happens within the update function at `contracts/oracle/src/contract.rs#L51`.

Recommendation

It's recommended to streamline the process by eliminating the redundant ADMIN check when updating the oracle configuration. Maintaining only one check either in `contracts/oracle/src/contract.rs#L44` or within the update function at `contracts/oracle/src/contract.rs#L51` should suffice, improving the code's efficiency and readability.

Status

The issue has been fixed in 8b78cfac971ff5e7af90f486a732016f955f8420.

Redundant string conversion

Low | **RES-BLEND-PRO17** | Gas Optimization | **Acknowledged**

Code Section

- contracts/fund-factory/src/handlers/save_router.rs#L54
- contracts/fund-factory/src/handlers/save_router.rs#L55

Description

The function `get_address_of_instantiated_contract`, when called, inside the `save_address` method, returns a string. This string is then transformed into the type `Addr` and stored in the Cache. However, in the `save_router_address` method, the addresses of both `market_maker` and `money_maker` are once again converted to string format.

Recommendation

It is recommended to streamline the handling of address types within the code. Since the function `get_address_of_instantiated_contract` returns a string that is transformed into type `Addr` and then reconverted to a string in the `save_router_address` method, it may be more efficient to standardize the format. Consider either retaining the string type throughout or performing the conversion at a single point to minimize unnecessary transformations between the string and `Addr` types. This consistency can lead to more maintainable and efficient code.

Status

The issue was acknowledged by Blend's team. The development team stated "To our understanding it is bad practice to rely on external validation. As contracts are independent units, they should be internally consistent. In the context of an application that leads to slight redundancy, at the benefit of removing a common and hard-to-find source for errors.".

Queries might reach gas limit

Info | **RES-BLEND-PRO18** | Denial of Service | **Acknowledged**

Code Section

- contracts/fund-factory/src/contract.rs

Description

The `get_funds` query in `contracts/fund-factory/src/contract.rs` might run out of gas in an execution context if too many Funds are stored.

The severity of this finding has been downgraded due to the RES-03 finding.

Recommendation

It is recommended to implement a pagination mechanism for queries.

Status

The issue was acknowledged by Blend's team. The development team stated "We only have one fund currently, so not an issue for now. Will be addressed in the full v1 release".

Events are not emitted when executing essential operations

Info | **RES-BLEND-PRO19** | Code Quality | **Resolved**

Code Section

- Not specified

Description

Throughout the entire codebase, it has been observed that the responses lack the necessary events.

Recommendation

It is advised to generate appropriate events when successful operations occur within the protocol. These events can provide transparency, aid in debugging, and enable integrations with other smart contracts or off-chain monitoring tools.

Status

The issue has been fixed in 8a0f27c4b7865a94f58d94338659fa6587bf2c0f.

Redundant addr validation

Info | **RES-BLEND-PRO20** | Gas Optimization | **Acknowledged**

Code Section

- Not specified

Description

Throughout the entire codebase, redundant address validation is taking place. Addresses are first verified within the fund-factory contract, and subsequently, they are validated again during the instantiation of other contracts within the protocol.

Recommendation

It is recommended to perform address validation only once if the contracts will not be instantiated outside of the funds-factory. This can help in eliminating unnecessary redundancy in the code.

Status

The issue was acknowledged by Blend's team. The development team stated "To my understanding it is bad practice to rely on external validation. As contracts are independent units, they should be internally consistent. In the context of an application that leads to slight redundancy, at the benefit of removing a common and hard-to-find source for errors.".

Lack of Error Handling

Info | **RES-BLEND-PRO21** | Code Quality | **Acknowledged**

Code Section

- Not specified

Description

The protocol's error handling predominantly depends on `StdError` and panicking macros. Unfortunately, these don't provide users with descriptive error messages, making it challenging for them to discern the root cause of issues.

Recommendation

It is recommended to use Custom Error Definitions Instead of relying solely on StdError and panicking macros. This allows for more descriptive error messages tailored to the context in which the error occurred.

Status

The issue was acknowledged by Blend's team. The development team stated "We will not further address this issue. This is of theoretical concern as the factory assigns the fund to the bow fund, so won't affect the protocol.".

Two-step ownership transfer is missing

Info | **RES-BLEND-PRO22** | Business Logic | **Resolved**

Code Section

- Not specified

Description

The contracts within the scope lack a two-step ownership transfer mechanism. Implementing such a process would enable the current owner to nominate a new owner. Subsequently, the account identified as the new owner could invoke a specific function, allowing them to assume ownership and carry out the necessary configuration update.

Recommendation

It is recommended to introduce a two-step ownership transfer process, structured as follows:

- The existing owner suggests a new owner's address, which undergoes validation.
- The account designated as the new owner asserts ownership, effectuating the configuration alterations.

Status

The issue has been fixed in 85eb8b5c2cc9982159c9b1a5d37c8afaf0f15a25.

Confusing function naming

Info | RES-BLEND-PRO23 | Code Quality | Acknowledged

Code Section

- packages/blend/src/authorized.rs#L48

Description

Confusing function name appears in:

- packages/blend/src/authorized.rs#L48

This could potentially lead to confusion and misinterpretation among developers, impeding the debugging process and potentially impacting the system's functionality.

Recommendation

It is recommended to address this issue by adopting a clear and consistent naming convention that accurately reflects the function purpose and the context in which it is used.

Status

The issue was acknowledged by Blend's team. The development team stated "Not going to address".

Confusing variable naming

Info | **RES-BLEND-PRO24** | Code Quality | **Acknowledged**

Code Section

- contracts/fin/src/contract.rs#L126

Description

Confusing variable name appears in:

- contracts/fin/src/contract.rs#L126

This could potentially lead to confusion and misinterpretation among developers, impeding the debugging process and potentially impacting the system's functionality.

Recommendation

It is recommended to address this issue by adopting a clear and consistent naming convention that accurately reflects the variable purpose and the context in which it is used.

Status

The issue was acknowledged by Blend's team. The development team stated "Not going to address".

Usage of nonconventional CosmWasm msg

Info | **RES-BLEND-PRO25** | Code Quality | **Acknowledged**

Code Section

- Not specified

Description

Our investigation has uncovered a noteworthy finding in the messaging protocol. Instead of using the standard syntax for messages, the protocol is employing nonconventional formats. As a case in point, consider the following message structure:

```
{execute:{deposit:  {debt_denom:  demo_denom }}}
```

This is unusual as the typical message would have been:

```
{deposit:  {debt_denom:  demo_denom }}.
```

This deviation from the conventional message structure may introduce complications for users and developers, and potentially for systems trying to interpret the message.

Recommendation

It is recommended to use conventional CosmWasm Messages.

Status

The issue was acknowledged by Blend's team. The development team stated "The suggested changes don't work as that removes the flexibility from passing an enum.".

Usage unwrap_or_default

Info | RES-BLEND-PRO26 | Code Quality | Resolved

Code Section

- contracts/fund-router/src/query.rs#L122
- contracts/fund-router/src/execute.rs#L203
- contracts/fund-router/src/execute.rs#L440

Description

The `unwrap_or_default` method is found in the codebase and may prove useful in a testing environment. However, should an error occur, a default value will be employed in its place. This could conceal the error, possibly resulting in behavior that is not as intended.

Recommendation

It is advised to eliminate the use of `unwrap_or_default` in the code and instead opt for explicit error handling to manage potential issues properly.

Status

The issue has been fixed in e9d765d5fc28d41d4dfcbf07a36161d429f00b20.

Lack of default slippage if not provided

Info | **RES-BLEND-PRO27** | Data Validation | **Acknowledged**

Code Section

- Not specified

Description

The deposit function empowers the `FUND` to deposit assets into Kujira Bow. Nonetheless, while slippage is an optional parameter, it lacks a predefined default value.

Recommendation

It's recommended to implement safe default values for slippage when none is supplied.

Status

The issue was acknowledged by Blend's team. The development team stated "This is feature, traders don't like being restricted from doing things they actually want to do and have a good reason for, that includes trading with slippage (i.e. during highly volatile times, incl. possibly network congestion it may be better to dump or buy at all cost instead of trying to get a transaction through while increasing slippage).".

Proof of Concepts

RES-01 Non-Withdrawable Surplus Asset in ghost

```
import { tx, registry } from "kujira.js";
import { DirectSecp256k1HdWallet } from "@cosmjs/proto-signing";
import { CosmWasmClient, } from "@cosmjs/cosmwasm-stargate";
import { coins, SigningStargateClient, GasPrice } from "@cosmjs/stargate";
import { readFileSync } from 'fs';


const RPC_ENDPOINT = "https://kujira-testnet-rpc.polkachu.com";


const MNEMONIC = "prize social visit floor crumble humble cabbage wink gap oblige
↪  ridge eyebrow";

const signer = await DirectSecp256k1HdWallet.fromMnemonic(MNEMONIC, {
  prefix: "kujira",
});

const [account] = await signer.getAccounts();

const client = await SigningStargateClient.connectWithSigner(
  RPC_ENDPOINT,
  signer,
  {
    registry,
    gasPrice: GasPrice.fromString("0.00125ukuji"),
  }
);



///Upload contract
const ghost_wasm = await readFileSync('../blend-rs/artifacts/ghost.wasm');
const ghost_msg = tx.wasm.msgStoreCode({sender:
↪  account.address,wasm_byte_code:ghost_wasm });
console.log(ghost_msg)
const upload_result = await client.signAndBroadcast(account.address, [ghost_msg],
↪  "auto");
console.log(upload_result)

const log = JSON.parse(upload_result.rawLog);
let codeId = null;

for (const event of log[0].events) {
  if (event.type === 'store_code') {
    for (const attribute of event.attributes) {
      if (attribute.key === 'code_id') {
        codeId = attribute.value;
```

```
        break;
      }
    }
  }
  if (codeId) break;
}

console.log(codeId);


const instantiate_ghos = tx.wasm.msgInstantiateContract({
  sender: account.address,
  admin: account.address,
  code_id: codeId,
  label:"ghost",
  msg:Buffer.from(JSON.stringify({ admin:account.address, ghost_markets:
  ↪  ["kujira1t88e60depax2zz43mt5ggxdzqe225guhn0jdyl3ccq5x30lcu2hqcnzt2t"] }))
})
const instantiate_result = await client.signAndBroadcast(account.address,
↪  [instantiate_ghos], "auto");


const log1 = JSON.parse(instantiate_result.rawLog);
let ghost_addr = null;

for (const event of log1[0].events) {
  if (event.type === 'instantiate') {
    for (const attribute of event.attributes) {
      if (attribute.key === '_contract_address') {
        ghost_addr = attribute.value;
        break;
      }
    }
  }
  if (ghost_addr) break;
}

console.log(ghost_addr);


const demo_denom = "factory/kujira1ltvwg69sw3c5z99c6rr08hal7v0kdzfxz07yj5/demo"
const usk_denom =
↪  "factory/kujira1r85reqy6h0lu02vyz0hnzhv5whsns55gdt4w0d7ft87utzk7u0wqr4ssll/uusk"

// ADD FUND ADDR
let msg_fund = tx.wasm.msgExecuteContract({
  sender: account.address,
  contract: ghost_addr,
  msg: Buffer.from(JSON.stringify({ update:{fund:account.address}})),});
  console.log(await client.signAndBroadcast(account.address, [msg_fund], "auto"));

// DEPOSIT TO GHOST
```

```
let msg_deposit = tx.wasm.msgExecuteContract({
  sender: account.address,
  contract: ghost_addr,
  msg: Buffer.from(JSON.stringify({ execute:{deposit: {debt_denom:
  ↪  "factory/kujira1r85reqy6h0lu02vyz0hnzhv5whsns55gdt4w0d7ft87utzk7u0wqr4ssll/uusk"
  ↪  } }})),

  funds: coins(4930092, demo_denom),
});
console.log(await client.signAndBroadcast(account.address, [msg_deposit], "auto"));
// BORROW FROM GHOST

const msg_borrow = tx.wasm.msgExecuteContract({
  sender: account.address,
  contract: ghost_addr,
  msg: Buffer.from(JSON.stringify({
    execute: {
      borrow: {
        amount: {
          denom: usk_denom,
          amount: "34335156"
        },
        collateral_denom: demo_denom
      }
    }
  })),
});
console.log(await client.signAndBroadcast(account.address, [msg_borrow], "auto"));

console.log(await client.getBalance(ghost_addr, usk_denom));

// REPAY
const msg_reply = tx.wasm.msgExecuteContract({
  sender: account.address,
  contract: ghost_addr,
  msg: Buffer.from(JSON.stringify({
    execute: {
      repay: {
        collateral_denom: demo_denom
      }
    }
  })),
  funds: coins(44335156,
  ↪  "factory/kujira1r85reqy6h0lu02vyz0hnzhv5whsns55gdt4w0d7ft87utzk7u0wqr4ssll/uusk")
});

console.log(await client.signAndBroadcast(account.address, [msg_reply], "auto"));

console.log(await client.getBalance(ghost_addr, usk_denom));
```

RES-02 Potential Panic during deposit_funds function Invocation in funds-router

```
#[test]
fn test_supply_is_not_zero_and_value_is_zero() {
    let fund_amount = Uint128::zero();
    let exchange_rate = Decimal::from_ratio(2u64, 1u64);
    let mut deps = mock_dependencies_with_oracle_res(
        &[],
        vec![(ETF_DENOM_1.to_string(), exchange_rate)],
        coin(fund_amount.u128(), ETF_DENOM_1),
        200u128.into()
    );

    let env = mock_env();
    let in_amount: u128 = 1000u128;
    let funds = vec![coin(in_amount, ETF_DENOM_1)];
    let info = mock_info(USER, &vec![]);
    let msg = get_instantiate_msg(TOKEN_NAME, ETF_DENOM_1);

    instantiate(deps.as_mut(), env.clone(), info.clone(), msg).unwrap();
    assign_fund(deps.as_mut(), env.clone(), info.clone());
    let msg = ExecuteMsg::Deposit {};
    let msg_info = mock_info(USER, &funds);

    execute(deps.as_mut(), env.clone(), msg_info, msg);
}

pub fn mock_dependencies_with_oracle_res(
    balances: &[(&str, &[Coin])],
    rates: Vec<(String, Decimal)>,
    fund_token: Coin,
    supply: Uint128,
) -> OwnedDeps<MockStorage, MockApi, MockQuerier<KujiraQuery>, KujiraQuery> {
    let balance_map: HashMap<_, _> = balances
    .iter()
    .map(|(s, c)| (s.to_string(), c.to_vec()))
    .collect();

    let supplies = calculate_supplies(&balance_map);
    let mut querier = MockQuerier::<KujiraQuery>::new(&[]).with_custom_handler(

    move |req: &KujiraQuery| -> MockQuerierCustomHandlerResult {
        match req {
            KujiraQuery::Bank(BankQuery::Supply { denom }) => {
                SystemResult::Ok(ContractResult::Ok(
                    to_binary(&SupplyResponse {
                        amount: coin(
                            supplies
                            .get(&denom.to_string())
                            .unwrap_or(&supply)//SUPPLY NASZEGO COINA
                            .u128(),
                            denom.to_string(),
                        ),
```

```
                    })
                    .unwrap(),
                ))
            }
            _ => SystemResult::Err(SystemError::UnsupportedRequest {
                kind: "not implemented".to_string(),
            }),
        }
    },
);

querier.update_wasm(move |req| -> MockQuerierCustomHandlerResult {
    match req {
        WasmQuery::Smart { contract_addr, msg } => {
            if contract_addr == "oracle" {
                let msg: OracleQueryMsg = from_binary(msg).unwrap();
                match msg {
                    OracleQueryMsg::Price { coin } => {
                        let rate = rates.iter().find(|&(d, _)| d == &coin.denom);
                        if let Some(&(_, rate)) = rate {
                            return SystemResult::Ok(ContractResult::Ok(
                                to_binary(&OraclePriceResponse {
                                    price: Coin {
                                        denom: USD.to_string(),
                                        amount: coin.amount * rate,
                                    },
                                })
                                .unwrap(),
                            ));
                        }
                        return SystemResult::Err(SystemError::UnsupportedRequest {
                            kind: format!("no exchange rate available for {}",
↪   coin.denom),
                        });
                    }
                    _ => {
                        return SystemResult::Err(SystemError::UnsupportedRequest {
                            kind: format!("query not implemented for {}",
↪   contract_addr),
                        })
                    }
                }
            }

            if msg == &to_binary(&FundQueryMsg::Positions {}).unwrap() {
                SystemResult::Ok(ContractResult::Ok(
                    to_binary(&FundPositionsResponse {
                        tokens: vec![fund_token.clone()],
                        market_maker: vec![],
                        money_market: vec![],
                    })
                    .unwrap(),
```

```
                ))
            } else {
                SystemResult::Err(SystemError::UnsupportedRequest {
                    kind: format!("query not implemented for {}", contract_addr),
                })
            }
        }
        _ => SystemResult::Err(SystemError::UnsupportedRequest {
            kind: "not implemented".into(),
        }),
    }
});

OwnedDeps {
    storage: MockStorage::default(),
    api: MockApi::default(),
    querier,
    custom_query_type: PhantomData,
}
```

RES-03 Denom order dependecy

```
#[test]
fn test_integration() {
    let mut deps = mock_dependencies();
    let msg = BowInstantiateMsg {
        admin: "admin".to_string(),
        bow: vec![],
        bow_staking: "admin".to_string(),
        claim_denoms: vec![],
    };

    let env = mock_env();
    let info = mock_info("admin", &[]);
    assert!(instantiate(deps.as_mut(), env.clone(), info.clone(), msg).is_ok());

    //MOCK QUERY RESPONSE
    deps.querier.update_wasm(move |req| -> MockQuerierCustomHandlerResult {
        match req {
            WasmQuery::Smart { contract_addr, msg } => {
                if contract_addr ==
↪   "kujira19kxd9sqk09zlzqfykk7tzyf70hl009hkekufq8q0ud90ejtqvvxs8xg5cq" {
                    let response = MMConfigResponse { owner:
↪   Addr::unchecked("input"), denoms:
↪   ["ukuji".into(),"factory/kujira1ltvwg69sw3c5z99c6rr08hal7v0kdzfxz07yj5/demo".into(),],
↪   price_precision: kujira::Precision::SignificantFigures(8u8) , decimal_delta:
↪   8i8, fin_contract: Addr::unchecked("input"), intervals: vec![Decimal::one()],
↪   fee: Decimal::one(), amp: Decimal::one()};
                    return SystemResult::Ok(ContractResult::Ok(
                        to_binary(&response)
                        .unwrap(),
                    ).into());
```

```
                } else if contract_addr == "bow_addr2" {
                    let response = MMConfigResponse { owner:
↪ Addr::unchecked("input"), denoms: ["denom1".into(), "denom3".into()],
↪ price_precision: kujira::Precision::SignificantFigures(8u8) , decimal_delta:
↪ 8i8, fin_contract: Addr::unchecked("input"), intervals: vec![Decimal::one()],
↪ fee: Decimal::one(), amp: Decimal::one()};
                    return SystemResult::Ok(ContractResult::Ok(
                        to_binary(&response)
                        .unwrap(),
                    ).into());
                } else {
                    let response = MMConfigResponse { owner:
↪ Addr::unchecked("input"), denoms: ["denom1".into(), "denom3".into()],
↪ price_precision: kujira::Precision::SignificantFigures(8u8) , decimal_delta:
↪ 8i8, fin_contract: Addr::unchecked("input"), intervals: vec![Decimal::one()],
↪ fee: Decimal::one(), amp: Decimal::one()};
                    return SystemResult::Ok(ContractResult::Ok(
                        to_binary(&response)
                        .unwrap(),
                    ).into());
                }
            }
            _ => SystemResult::Err(SystemError::UnsupportedRequest {
                kind: "not implemented".into(),
            }),
        }
    });

    let info = mock_info("admin", &[]);
    let msg = BowExecuteMsg::Update(BowUpdate { fund: Some("fund".to_string()), bow:
↪ Some(vec!["kujira19kxd9sqk09zlzqfykk7tzyf70hl009hkekufq8q0ud90ejtqvvxs8xg5cq".to_string()]
↪ bow_staking:
↪ Some("kujira1e7hxytqdg6v05f8ev3wrfcm5ecu3qyhl7y4ga73z76yuufnlk2rqd4uwf4".to_string()),
↪ claim_denoms: Some(vec![]), });
    let execute_response = execute(deps.as_mut(), env.clone(), info, msg).unwrap();

    println!("Execute update: {:?}", execute_response);

    //Simulate the deposit!
    let info = mock_info("fund", &[coin(1_000_000,
↪ &"factory/kujira1ltvwg69sw3c5z99c6rr08hal7v0kdzfxz07yj5/demo".to_string()),coin(1_000_000,
↪ &"ukuji".to_string())]);
    let msg = BowExecuteMsg::Execute(MarketMakerExecuteMsg::Deposit { max_slippage:
↪ None });
    let execute_response = execute(deps.as_mut(), env.clone(), info,
↪ msg.clone()).unwrap_err();

    assert_eq!(execute_response, ContractError::Std(StdError::NotFound { kind:
↪ "cosmwasm_std::addresses::Addr".to_string() }));
}
```